Reinforcement Learning 2024 Assignment 3: Policy-based Reinforcement Learning

Sherry Usman, Qin Zhipei, Meghan[1]

Abstract

In this paper we explored Policy-based Reinforcement Learning as a paradigm of reinforcement learning architectures and the various algorithms within it. The environment we focused on is one of the OpenAI Gym Libraries - Lunar Lander. We implemented algorithms such as REINFORCE, Actor-Critic and their specifications and also delve further into the effects of bootstrapping and baseline subtraction on actor-critic networks. We found that Actor-Critic with bootstrapping and baseline subtraction performs the best. Furthermore, we found that learning rates have a significant effect on performance here as well with a learning rate of 0.001 and 0.01 performing much better than a learning rate of 0.1. We also compared and contrasted the performances of the algorithms when the architecture of the neural network is varied, specifically the number of hidden layers and number of neurons. We also analyze the effects of entropy regularization on these algorithms and found that algorithms perform better with entropy regularisation than without.

1. Introduction

In this paper we will delve into the Lunar Lander environment taken from the OpenAI Gym library. The Lunar Lander game is a classical reinforcement learning environment, based on the physics enginge box2d (Brockman et al., 2016) where the goal is to optimise the trajectory of a rocket such that it lands between the two flags. While a number of different techniques can be used to solve this problem, our paper specifically concentrates on the use of **REINFORCE** and **Actor-Critic**. Through this paper, we hope to provide the specifications of various Policy Gradient methods while also including the tried and tested hyper-parameters that produce the best results.

The Lunar Lander environment poses a number of unique challenges compared to the environments we have encountered previously. While the action space A is still discrete with four potential actions, the observation space is an 8-dimensional vector comprising of the positional coordinates of the lunar lander (x and y), its linear velocities in x and in y respectively, its angle, its angular velocities and two boolean values that represent whether each leg is in contact with the ground or not. The inclusion of a continuous action or state space makes tabular/value-based reinforcement learning algorithms quickly unfeasible as the cost of storing the values for each state-action pair makes our problem much more complex.

Thus, to tackle this problem we use Policy Gradient methods which can handle continuous state and action spaces much better, as they directly optimise the expected return and can provide continuous action distributions as output, making them also suitable for continuous action spaces.

Value	Action
0	Do nothing
1	fire left orientation engine
2	fire main engine
3	fire right orientation engine

Table 1. Action Space of Lunar Lander Environment

2. Framework

For our implementation of Policy Gradient Methods, we used the Python **Pytorch** package.

Our initial structure is as follows: an input layer of size 8 corresponding to the size of the observation space. This input layer feeds into a first hidden layer with 128 nodes. This then feeds into a hidden layer comprising of 128 nodes which finally outputs to a layer comprising of 1 node representing the action that should be taken. Since the Lunar Lander environment is also highly stochastic, to reduce noise in our graphs we computed different algorithms over 1000 episodes and averaged the results every 5 episodes to capture the general trends.

Furthermore, we are aware that total rewards is not always the best parameter to measure algorithm accuracy as the number of timesteps in each episode may differ and lead to exploding or shrinking rewards. Thus, in some cases we also include another parameter to measure accuracy which

is the number of timesteps.

3. Policy-Based Reinforcement Learning

The policies we looked at in the previous papers were *value-based*. This means that they looked at state-value pairs in the environment and calculated the rewards of such pairs using different exploration strategies like Boltzmann or Epsilon-Greedy. Policy based reinforcement learning methods differ from value-based method as they do not utilize a value function to determine the next possible action. Used primarily in continuous action space RL environments, policy based reinforcement learning methods use a parameterized policy function π_θ where θ represents the parameters of the function.

During training the agent iteratively interacts with the environment and after a number of episodes the collected data known as the trajectory τ is evaluated to understand the performance of the policy.

- If τ yields high rewards then the parameters θ are adjusted in such a way that the likelihood of taking similar actions as those in τ is increased.
- On the other hand if the trajectory τ yields lower rewards then θ is adjusted in such a way that the likelihood of taking similar actions as those in τ is decreased.

This process of iteratively evaluating the trajectories and updating the parameters according to the obtained rewards/losses helps to refine the policy to maximize cumulative rewards in the long run (Plaat, 2022).

Policy Gradient methods pose a number of advantages that make them suitable algorithms for reinforcement learning environments. Firstly since Policy Gradient methods directly optimise a policy function rather than approximating the value of each state-action pair they tend to be much more stable and robust in high-dimensional environments like Lunar Lander, allowing more stable convergence after a number of training episodes. Additionally, since they do not need state-action pairs to approximate values they do not run the risk of getting stuck in local optima like value-based methods and can approximate complex non-linear relationships more accurately than value-based algorithms. Lastly, policy gradient methods tend to be more sample efficient as they need less existing data to achieve a good performance and can even approximate rewards in areas with sparse reward signals.

There are a number of different policy-based reinforcement learning algorithms. However for this paper, we are limiting the scope of our research to the algorithms listed below:

- REINFORCE
- Actor-Critic with Bootstrapping
- Actor-Critic with Baseline Subtraction
- Actor Critic with Bootstrapping and Baseline Subtraction

These algorithms are discussed in further sections.

For any reinforcement learning algorithm it is necessary to decide on an action selection policy. Traditional action selection algorithms used before such as Boltzmann or E-greedy cannot be used in this paper as they are better suited for value-based reinforcement learning algorithms working with a discrete action space and/or state space. This is because both these algorithms scans potential actions that can be taken by the agent at a particular state and calculates their value, which helps inform the agent's next step. This is not possible in Policy Gradient methods which are less concerned with state-action pairs and more with the policy function.

Hence, in this paper we work with **Entropy Regularisation**. Entropy regularisation is an exploration technique that helps balance the exploration to exploitation ratio by encouraging the agent to explore more diverse actions and unfamiliar spaces. It does this by increasing by adding an entropy term to the loss function to promote action diversity and preventing sub-optimising to local rewards. The entropy term $H(X)$ is defined in the equation below.

$$H(X) = -\sum \pi(x) log(\pi(x))$$

where

- x is the set of possible actions
- $\pi(x)$ is a function representing the probability of taking an action x

3.1. REINFORCE

REINFORCE is a commonly used Policy-based Reinforcement Learning algorithm which is also termed as a classic Monte-Carlo Policy Gradient algorithm. It was first introduced in 1992 by Ronald J. Williams with an aim to maximise expected cumulative rewards by adjusting the policy parameters (Williams, 1992). It does this by learning from the agent trajectory (including the visited states, executed actions and incurred rewards) and estimating gradients. Because it needs a full trajectory in order to construct a sample space it is updated as an off-policy algorithm (Richard S. Sutton, 2018).

$$\pi_\theta(a, s) = Pr(A_t = a|St = s, \theta_t = \theta) \quad (1)$$

The equation above helps us recall the policy function $\pi_\theta(a, s)$ which indicates the probability of taking an action a in the state s with the parameters θ. We can also

define a trajectory τ as the sum of rewards when following a particular path. This is shown in the equation below.

$$R(\tau) = [\sum_{t=0}^{T-1} r_t] \tag{2}$$

The REINFORCE algorithm optimises the policy function $\pi_\theta(a, s)$ by fine-tuning its parameters θ such that it increases the likelihood of actions in trajectories τ which lead to higher cumulative rewards. It does this by calculating the total expected rewards J(θ) with respect to a particular parameter value θ and adjusting θ accordingly. Thus, J(θ) is optimised as shown in equation 2.

$$\nabla J(\theta) = E_{\pi_\theta}[R(\tau)] = E_{\pi_\theta}(\sum_{t=0}^{T-1} \nabla log \pi_\theta(a_t s_t) R(\tau)] \tag{3}$$

where $\nabla J(\theta)$ represents the gradient in the expected return function J(θ), Eπ_θ denotes the expectation over trajectories R sampled under policy π_θ, R_τ indicates the rewards obtained from the trajectory τ and π_θ indicates the policy function with parameters θ.

The algorithm for REINFORCE can thus be defined as follows.

Algorithm 1 REINFORCE Algorithm

Data: parameter θ, policy function π_θ, maximum time-steps T, number of episodes E, learning$_r ate \alpha$
Result: Selected action
Initialise θ arbitrarily
for $e = 1$ **to** E **do**
  Initialise state s **for** $t = 1$ **to** $T - 1$ **do**
    Sample action a from π_θ
    Execute action a and get the next state s_{t+1} and reward r_t
    $\theta \leftarrow \theta + \alpha \cdot \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)$;
  end
end
Return parameter θ

Figure 2 and 3 shows our results for REINFORCE without entropy regularisation. While REINFORCE is touted as a simpler Policy Gradient method it can often suffer from high variances causing slower convergence to optimal values because it requires an entire episode to be sampled. This is very visible from the results. Even after an average results of over 1000 episodes, we are unable to see a dramatic rise in learning as we expected. However, we can see the effect of learning rate on model performance as learning rates of 0.001 and 0.01 outperform learning rate of 0.1.

To improve our existing results we decided to implement a version of REINFORCE that includes entropy regularisation

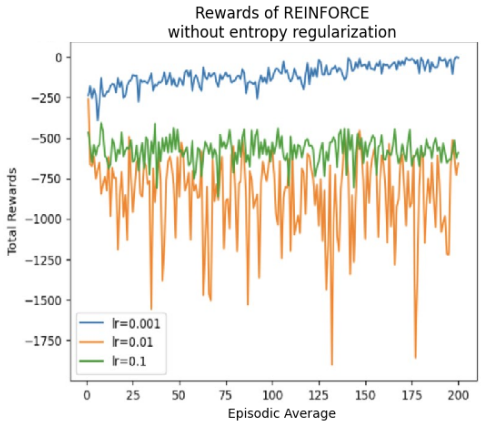

Figure 1. The figure above shows the rewards of REINFORCE without entropy regularisation with different learning rates. As we can see, almost all of learning rates show a rising trend of rewards, with the graph for learning rates 0.01 and 0.001 showing the most promise.

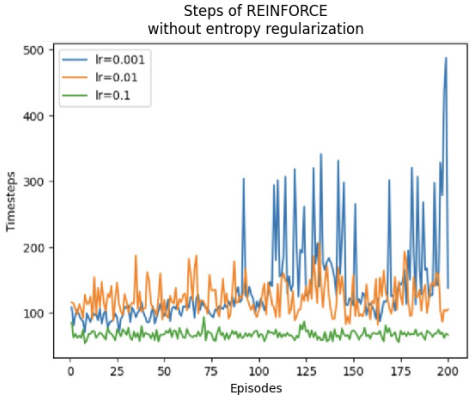

Figure 2. The figure above shows the steps of REINFORCE without entropy regularisation with different learning rates. As we can see, almost learning rates show a rising trend of steps, with the graph for learning rates 0.01 and 0.001 showing the most promise.

and as seen from our figure 4, we reach higher cumulative rewards, touching 100 and there is more stability in our results.This is because the inclusion of an entropy term to the loss function helps penalise trajectories with lesser randomness (or entropy) and thus encourages the policy network to explore more diverse state and action spaces, increasing exploration and preventing getting stuck in local maxima.

3.2. Actor-Critic Methods

In the REINFORCE algorithm, we did not observe the fast convergence to optimal values as we hoped. This was probably because in REINFORCE algorithms, the entire episode

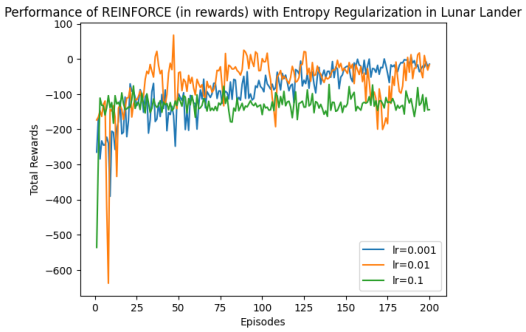

Figure 3. The figure above shows the rewards of REINFORCE with entropy regularisation with different learning rates. As we can see, all learning rates show a rising trend of rewards, with the graph for learning rates 0.01 and 0.001 showing the most improvement and the learning rate 0.1 stabilising but not showing much improvement.

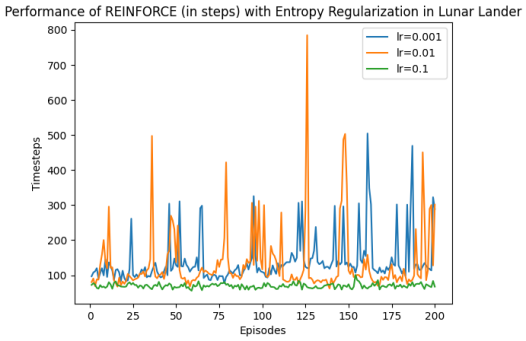

Figure 4. The figure above shows the total steps of REINFORCE with entropy regularisation with different learning rates. As we can see, all learning rates show a rising trend of rewards, with the graph for learning rates 0.01 and 0.001 showing the most improvement and the learning rate 0.1 stabilising but not showing much improvement.

must be sampled and thus suffers from high variance. In this section we will discuss Actor Critic methods, which provide a solution to combat high variance by combining the advantages of both value-based and policy-based methods and achieve optimal converging rewards.

Actor Critic methods are Temporal Difference methods and have a separate memory structure to store the policy function independent of the value function. They generally have two neural networks: a policy structure known as an *actor* and an estimated value function known as the *critic*. While the policy function (or actor) returns a probability distribution of actions that can be taken in specific states, a value function (or the critic) determines the expected reward gained by following the policy. (Grondman et al., 2012). Thus the value function is called a critic as it criticises the actions taken by the actor in the form of a TD error. This scalar signal indicates whether the actions taken by the agent are better or worse than what the critic expected.

The TD error can be evaluated as follows in the following equation

$$\delta_t = R_{t+1} + \gamma * V_t(S_{t+1}) - V_t(S_t)$$

where

- δ_t is the temporal difference error at timestep t
- R_{t+1} is the reward for transitioning from state S_t to the state S_{t+1}
- γ is the discount factor which discounts future rewards
- $V_t(S_{t+1})$ is the estimated value of going to state S_{t+1} at timestep t according to the critic. This represents the estimated cumulative rewards that can be gained from step S_{t+1} onwards.
- $V_t(S_t)$ is the estimated value of going to state S_t according to the critic. This represents the estimated total rewards that can be gained from step S_t onwards.

Thus, a positive TD error ($\delta_t > 0$) indicates that the value of transitioning from state S_t to S_{t+1} is greater than expected and thus the action responsible should be taken more often. On the other hand a negative TD error ($\delta_t < 0$) indicates that the value of transitioning from state S_t to S_{t+1} is lesser than expected and thus the action responsible should be avoided in the future. Thus, through this iterative process of updating the TD error the Actor Critic algorithm tends to strike a balance between exploration and exploitation. The actor explores the state space and taking more diverse actions and the critic evaluating the consequences of the actions. When the actions are more risky and produce lesser expected returns the TD error is negative and when they are more fruitful and produce higher expected returns the TD error is positive.

3.2.1. Bootstrapping

A variation of Actor-Critic incorporates a bootstrapping technique which combines elements of both policy-based and value-based methods. In this architecture the actor, as usual, learns a parameterized policy function π_θ that maps states to actions. The critic uses bootstrapping to update the value function and estimate the value of the state-action pairs. Thus, instead of waiting for the cumulative rewards from the environment it updates the value function using its own estimates. By using bootstrapping, the Actor Critic method can use the Critic's immediate feedback to adjust the Actor's strategy, thus achieving a better balance between exploring the environment and learning optimized action strategies. The descent advantage loss allows the actor to choose the actions that outperform the baseline by minimising the difference between them, while the ascent policy

gradient drives policy updates to maximize the cumulative reward by increasing the likelihood of selecting actions with higher advantages.

The algorithm for Actor-Critic with Bootstrapping is given below (Plaat, 2022)

Algorithm 2 Actor-Critic with Bootstrapping

Data: parameter θ, policy function π_θ, maximum timesteps T, number of episodes E, estimation depth n, learning rate α, value function $V_\phi(s)$
Initialise θ and ϕ arbitrarily

Repeat
for $e \in 1$ **to** E **do**
  Initialise state s
  Sample trajectory $\tau = s_0, a_0, r_1, ..., s_T$ for $\pi_\theta(a|s)$
  for $t = 1$ **to** $T - 1$ **do**
    Compute cumulative reward $\hat{Q}_n(s_t, a_t)$ for the n-step target
    $\hat{Q}(s_t, a_t) = \sum_{k=0}^{n-1} \gamma^k \cdot r_{t+k} + \gamma^n \cdot V_\phi(s_{t+n})$
  end
end
$\phi \leftarrow \phi - \alpha \cdot \nabla_\phi \sum_t (\hat{Q}_n(s_t, a_t) - V_\phi(s_t))^2$
$\theta \leftarrow \theta + \alpha \sum_t [\hat{Q}_n(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t)]$
Until $\nabla_\theta J(\theta)$ converges below ϵ
Return parameter θ

Figures 6, 7 and 8 show us the performances of Actor-Critic with bootstrapping when varying different hyper-parameters such as learning rate, the number of hidden layers and the number of neurons. As seen in figure 6 learning rate of 0.005 shows a jump in average returns compared to other learning rates. Furthermore, we can also see in figure 7 that bootstrapping with 2 or 3 hidden layers tends to learn faster than bootstrapping with only one hidden layer. Lastly, figure 8 shows the effects of the number of neurons on learning rate. We can see that neural networks with 64 neurons learn faster than neural networks with 128 or 256 neurons. While the results do provide some key insights, it is important to note that when averaging rewards in highly stochastic environments can lead to final graphs where the overall increase is not as dramatic.

3.2.2. Baseline Subtraction

Actor-Critic algorithms could also employ baseline subtraction which is a technique used to reduce the variance and improve convergence. We recall the previous policy gradient function denoted by

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t]$$

By subtracting a baseline $b(s_t)$, typically a value function,

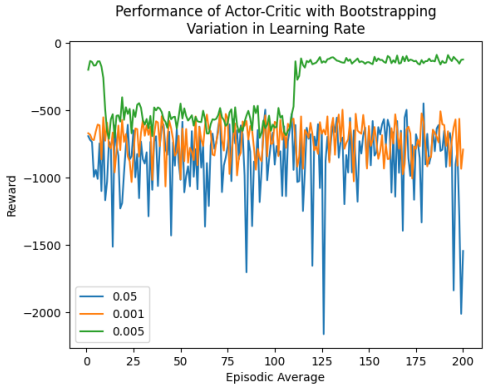

Figure 5. The figure shows the performance of Actor Critic with bootstrapping while varying learning rates. When the learning rate is set to 0.005, relatively good results are achieved in training the agent. However, after 1000 episodes (averaged over every 5 episodes for a total of 200 averages), the reward still remains below zero. Conversely, when the learning rate is set to a higher value (0.05) or a lower value (0.001), the agent does not train at all.

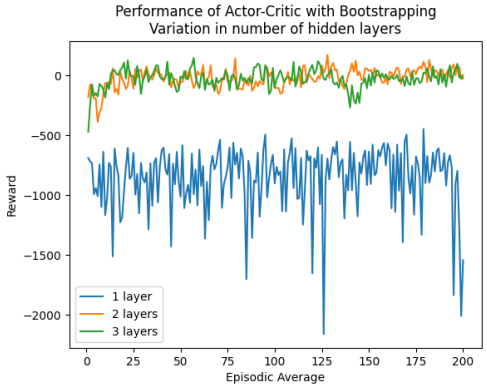

Figure 6. The figure shows the performance of Actor Critic with bootstrapping while varying the number of hidden layers. When both the policy network and the value network have only one hidden layer, the agent cannot be trained, as evidently seen in the graph. Increasing the hidden layers of these two networks can improve the training effect. The graph demonstrates the improvement in performance with more than one hidden layer, with a significantly better reward than with one hidden layer

from the observed cumulative reward helps reduce make them smaller and more stable. This reduces the variance, without changing the expectation. (Silver, 2020)

The updated policy gradient function after baseline subtraction is shown below.

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t - b(s_t)]$$

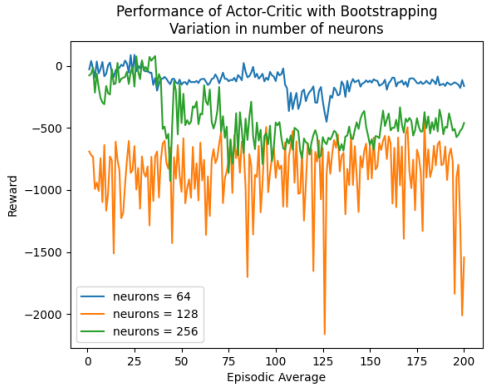

Figure 7. The result of Actor Critic with different neurons. According to results of different network layers, we set both the policy network and the value network to have two hidden layers. The results shows that networks with fewer neurons tend to perform relatively better, but there is still a long way to go before the agent is successfully trained.

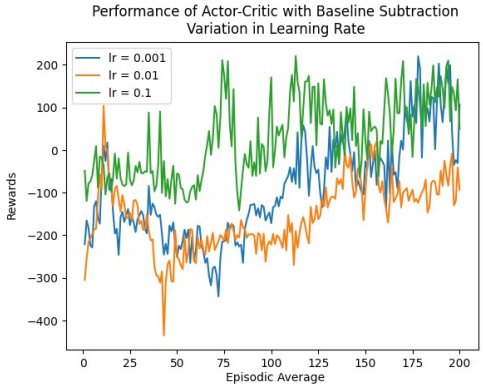

Figure 8. The result of Actor Critic with Baseline Subtraction with different learning rates. When evaluating three different learning rates (0.001, 0.01, 0.1), it was observed that the rate of 0.1 achieved the highest reward after averaging across 200 episodes. The learning rate of 0.001 initially shows slow progress but gradually improves over time. On the other hand, the learning rate of 0.01 results in the poorest performance.

Algorithm 3 Actor-Critic with Baseline Subtraction

Data: parameter θ, policy function π_θ, maximum timesteps T, number of episodes E, estimation depth n, learning rate α, value function $V_\phi(s)$
Result: Selected action
Initialise θ and ϕ arbitrarily
for $e = 1$ **to** E **do**
  Initialise state s
  Sample trajectory τ for π_θ
  for $t = 1$ **to** $T - 1$ **do**
    Compute advantage function using the cumulative reward and the value function; $\hat{A}_n(s_t, a_t) = \hat{Q}_n(s_t, a_t) - V_\phi(s_t)$
  end
  Compute descent advantage loss ϕ
  Compute ascent policy gradient θ
end
return parameter θ

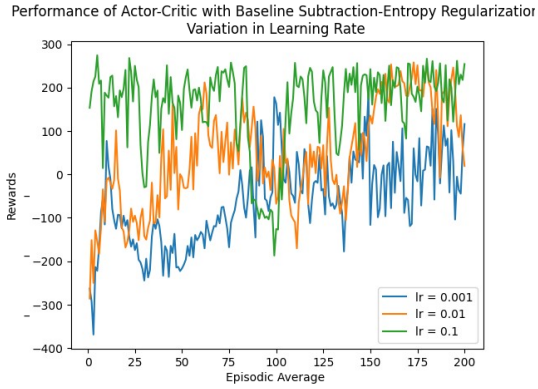

Figure 9. The result of Actor Critic with Baseline Subtraction and entropy regularization with different learning rates. When evaluating three different learning rates (0.001, 0.01, 0.1), it was observed that the rate of 0.1 achieved the highest reward after averaging across 200 episodes. The learning rate of 0.001 initially shows slow progress but gradually improves over time. On the other hand, the learning rate of 0.01 results in the poorest performance.

Figure 9 and Figure 10 compares the performances of Actor-Critic with Baseline Subtraction, with and without entropy regularization. Both agents were tested against learning rates of 0.001, 0.01 and 0.1. As seen from figure 9, the algorithm provided better returns than vanilla Actor-Critic algorithms. However, the rewards were lower than that of entropy regularization. In figure 10, it is evident that the returns are more (comparing the rewards of a learning rate of 0.1). However, a low learning rate such as 0.001 is not beneficial, as the gain is much slower than a learning rate of 0.1.

3.2.3. Bootstrapping and Baseline Subtraction

Certain Actor-Critic algorithms combine both bootstrapping and baseline subtraction. They combine the advantages of bootstrapping by using the Critic's feedback to adjust the Actor's strategy and by using baseline subtraction to reduce the variance and keep the rewards small, while maintaining stability. This is achieved by using both descent advantage loss and ascent policy gradient. The equations for Descent Advantage Loss and Ascent Policy Gradient are given below.

Descent Advantage Loss is calculated by

$$\theta \leftarrow \theta - \alpha * \nabla_\theta \Sigma_t (\hat{A}_n(s_t, a_t))^2$$

Ascent Policy Gradient is calculated by

$$\theta \leftarrow \theta + \alpha * \Sigma_t[\hat{A}_n(s_t, a_t) * \nabla_\theta \log \pi_\theta(a_t|s_t)]$$

Where

- θ is the parameter of the policy function
- α is the learning rate
- γ is the discount factor which discounts future rewards
- $\hat{A}_n(s_t, a_t)$ is the advantage function which represents the estimated advantage of taking an action a_t in the state s_t
- ∇_θ represents the gradient with respect to the policy parameters θ

Both descent advantage loss and ascent policy gradient uses the advantage function. The advantage function subtracts the value V from a state-action value estimate Q. This allows to estimate how better the action is, compared to the expectation at that state. This is expressed as

$$\hat{A}_n(s_t, a_t) = \hat{Q}_n(s_t, a_t) - V_\phi(s_t)$$

to estimate the cumulative reward

Algorithm 4 Actor-Critic with Bootstrapping and Baseline Subtraction

Data: parameter θ, policy function π_θ, maximum timesteps T, number of episodes E, estimation depth n, learning rate α, value function $V_\phi(s)$
Result: Selected action
Initialise θ and ϕ arbitrarily
for $e = 1$ **to** E **do**
  Initialise state s
  Sample trajectory τ for π_θ
  for $t = 1$ **to** $T - 1$ **do**
    Compute cumulative reward $\hat{Q}_n(s_t, a_t)$ for the n-step target
    Compute advantage function using the cumulative reward and the value function; $\hat{A}_n(s_t, a_t) = \hat{Q}_n(s_t, a_t) - V_\phi(s_t)$
  end
  Compute Descent Advantage Loss ϕ
  Compute Ascent Policy Gradient θ
end
return parameter θ

Figures 11, 12 and 13 shows the performances of Actor-Critic with Bootstrapping and Baseline Subtraction when various hyper-parameters are tuned such as learning rate, number of hidden layers and number of neurons. As seen in Figure 11, a learning rate of 0.1 provides better results when compared to learning rates of 0.001 and 0.01. The learning rate of 0.001 shows slower progress, but improves gradually.

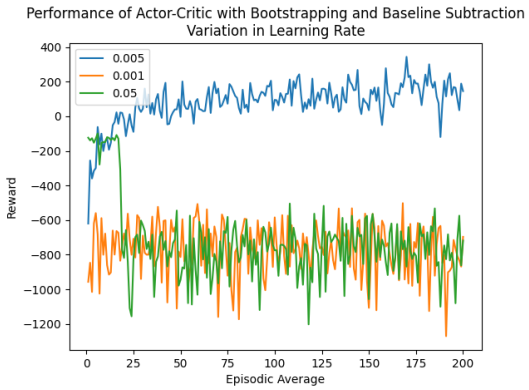

Figure 10. The figure above shows the rewards of Actor Critic with Bootstrapping and Baseline Subtraction with different learning rates. The results were averaged over every 5 episodes. As seen in the graph, the learning rate of 0.005 show a rising trend in rewards when compared to a learning rate of 0.001 and 0.05

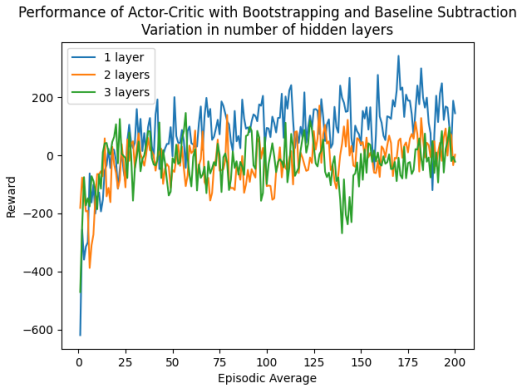

Figure 11. The results of Actor-Critic with Bootstrapping and Baseline Subtraction with variations in number of hidden layers of the policy network and the value network. The results were averaged over 2 episodes. The graph shows comparable performance with 1, 2 and 3 hidden layers, with 1 hidden layer performing slightly better than 2 and 3 hidden layers.

From figure 12, it is observed that the number of hidden layers did not play a significant role, surprisingly. All 3 variations performed similarly, with a variation of 1 and 2 hidden layers performing slightly better. The performance of Actor-Critic agents with bootstrapping and baseline subtraction was graphed out in Figure 13. Here, it is seen that 128 is the optimal number, with a steep improvement in performance, as it produced better results, when compared to 64 and 256 neurons.

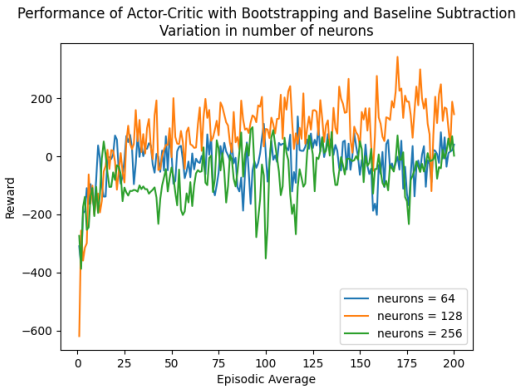

Figure 12. The results of Actor-Critic with Bootstrapping and Baseline Subtraction with variations in number of neurons of the policy network and the value network. The results show that the layers with 128 neurons provided better results, when compared to layers with 64 neurons and 256 neurons. When the number of neurons was set to 128, the agent achieved a better reward, reaching a peak value of over 200. In comparison, networks with 64 and 256 neurons achieved slightly poorer training results.

4. Goals Achieved

4.1. Effects of Policy Gradients of REINFORCE on Variance

As we can see from the results of our implementation of REINFORCE, Policy Gradient methods can have a de-stabilising effect on rewards, particularly in high-dimensional state spaces such as Lunar Lander. We can see this with the high amount of noise generated in our average reward and time step plots in figures 2 and 3. This may be because REINFORCE samples entire trajectories (series of states, actions and rewards) stochastically rather than observing the rewards gained from individual states or actions and thus there can be high variability in the returns produced by different trajectories which may only be differ by one action or state. Furthermore, since this environment is more complex than classical environments like CartPole, the probability distribution produced can have a high variance between different state

We can see that the implementation of Entropy Regularisation in REINFORCE helps to significantly reduce noise in REINFORCE. This can be seen by the difference in noise in Figure 2 and 4. This is because Entropy Regularisation can help the architecture balance the exploration-to-exploitation ratio, allowing the agent to explore more unfamiliar state spaces in some cases and prioritizing greedy actions in others.

4.2. Effects of Bootstrapping and Baseline Subtraction On Policy Gradients

While bootstrapping is done to reduce the variance of the policy gradient, we observe that bootstrapping itself is not sufficient to help the architecture achieve rewards that converge to a stable optima over time. This is evident from our results in figure 6, 7 and 8. This may be because while bootstrapping tries to bring the advantages of value-based learning to policy-based learning, it may end up doing the exact opposite in the form of higher instability when value estimates are updated based on unstable environment rewards by the critic. This often causes the instability to explode and propagate further. However, bootstrapping combined with baseline subtraction works much better than bootstrapping alone. This is shown by our positive results in 10, 11 and 12. This is because the subtraction of a baseline term from estimated advantages helps to compensate for the variance generated by bootstrapping, allowing the estimated advantages to be more stable and accurate, allowing faster convergence.

4.3. Comparison of Performance

First, we compared the results of the REINFORCE algorithm. When we used REINFORCE without entropy regularization, we tested three different learning rate settings, and it turns out that the smallest value (0.001) achieved the best results. However, when entropy regularization was added to the model, none of the three learning rates achieved equally good results.

For the Actor-Critic model, the results indicate that using only the bootstrapping technique does not lead to ideal outcomes. Employing only baseline subtraction can result in relatively more satisfactory rewards, but the best approach is to combine these two techniques, which can push the final reward to exceed 200. Comparing the best outcomes of REINFORCE and Actor-Critic clearly demonstrates the superiority of the Actor-Critic algorithm in our study.

4.4. Effect of entropy regularization on performance

As explained earlier, entropy regularization promotes action diversity. This is evidently seen in Figure 2 and Figure 4 for REINFORCE agent, particularly with a learning rate of 0.01. The REINFORCE agent with entropy regularization provides higher rewards, when compared to the agent without regularization. This is also seen in Figure 9 and Figure 10, actor-critic with baseline subtraction. The actor-critic agents with entropy regularization performed better with learning rates 0.01 and 0.1, than without entropy regularization. The rewards were consistently higher and the curve was more stable. This shows that entropy regularization influences policy optimization and stability of the agents.

References

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Ivo Grondman, Lucian Busoniu, Gabriel Lopes, and Robert Babuska. A survey of actor-critic reinforcement learning: standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics*, 2012. URL https://hal.science/hal-00756747/document.

Aske Plaat. *Deep Reinforcement Learning*. Springer Nature Singapore, 2022. ISBN 9789811906381. doi: 10.1007/978-981-19-0638-1. URL http://dx.doi.org/10.1007/978-981-19-0638-1.

Andrew G. Barto Richard S. Sutton. *Reinforcement Learning: An Introduction*. The MIT Press, 2018. URL https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf.

David Silver. Ucl course on reinforcement learning. Lecture Number 7, 2020. URL https://www.davidsilver.uk/wp-content/uploads/2020/03/pg.pdf.

Ronald J Williams. *Simple statistical gradient-following algorithms for connectionist reinforcement learning*. 1992. URL https://doi.org/10.1007/BF00992696.